NEW EDITION

DIGITAL COPY


ICS
Part 2

STAR

# Computer Science

## Practical Notebook

Khurram Arslan

# LIST OF PRACTICALS

**According to the National Curriculum on Computer Science for ICS-Part 2 (Class - XII)**

## Microsoft Access

| No. | Practical |
|---|---|
| 1. | Create different tables and assign primary key. |
| 2. | Create simple queries using wizard and design view. |
| 3. | Create relationship between tables. |
| 4. | Create simple forms using wizard and design view. |
| 5. | Create reports using wizard and design view. |
| 6. | Use of summary and calculated fields. |

## C Language

| No. | Practical |
|---|---|
| 1. | Writing a program which prints a text of 4 lines consisting of characters, integer values and floating point values using printf statement. |
| 2. | Writing a program that reads and prints the data using Escape Sequence, (Asking the name, age, height and gender of the student using scanf and printf statements). |
| 3. | Writing a program which uses operators (Calculate the area of triangles, volume of spheres and arrange the resultant values in ascending order). |
| 4. | Writing a program which uses 'for' loop statement (Generate the multiplication table from 2 to 20). |
| 5. | Writing a program which uses 'while' loop and nested 'while' loop (Use 'for' loop and continue the process in 'while' loop satisfying this condition. |
| 6. | Finding the factorial of N using 'while' loop, read value of N using scanf, and print the factorial of various N. |
| 7. | Draw a checkerboard and print it using if-else statement, and extend the program using nested if-else. |
| 8. | Writing a program which uses a 'switch' statement and breaks the program if certain condition is observed. Repeat the program with 'case' statement. |
| 9. | Writing a function which generates factorial of N and calls this function in the 'main' program. |
| 10. | Writing a program which uses multiple arguments in a function (Develop a user-defined function to generate a rectangle. Use the function for passing arguments to draw different sizes of rectangles and squares.) |

# Contents

part B

# C Language



Part B

# INTRODUCTION TO C

## The Origins of the C Language

Dennis Ritchie invented and first implemented the programming language C on a DEC PDP-11 that used the UNIX operating system. The language is the result of a development process that started with an older language called BCPL. Martin Richards developed BCPL, which influenced Ken Thompson's invention of a language called B, which led to the development of C in the 1970s.

## A Middle-Level Language

C is often called a middle-level computer language. This does not mean that C is less powerful, harder to use, or less developed than a high-level language such as BASIC or PASCAL; nor does it imply that C is similar to, or presents the problems associated with, assembly language. The definition of C as a middle-level language means that it combines elements of high-level languages with the functionalism of assembly language.

## A Structured Language

Although the term block-structured does not strictly apply to C, C is commonly called a structured language because of structural similarities to ALGOL, Pascal, and Modula-2. Technically, a block structured language permits procedures or functions to be declared inside other procedures or functions. In this way the concepts of "global" and "local" are expanded through the use of scope rules, which govern the "visibility" of a variable or procedure. Since C does not allow the creation of functions within functions, it is not really block structured.

## A Programmer's Language

One might respond to the statement, "C is a programmer's language" with the question, "Aren't all programming languages for programmers?" the answer is an unqualified "No!" Consider the classic examples of nonprogrammers's languages, COBOL and BASIC. COBOL was designed to enable nonprogrammers to read and, presumably, understand the program. BASIC was created essentially to allow nonprogrammers to program a computer to solve relatively simple problems.

In contrast, C stands almost alone in that it was created, influenced, and field-tested by real working programmers. The end result is that C gives the programmer what the programmer want: few restrictions, few complaints, block structures, stand alone functions, and a compact set of keywords. It is truly amazing that by using C, a programmer can achieve nearly the efficiency of assembly code, combined with the structure of ALGOL or Modula-2. It is no wonder that C is easily the most popular language among topflight professional programmers.

# COMPILERS VERSUS INTERPRETERS

## Compilers Versus Interpreters

The terms compiler and interpreter refer to the way in which a program is executed. In theory, any programming language can be either compiler or interpreted, but some languages are usually executed one way or the other. For example, BASIC is usually interpreted and C is usually compiled.

An interpreter read the source code of your program one line at a time and performs the specific instructions contained in that line. A compiler read the entire program and converts it into object code, which is s translation of the program source code in a form that can be directly executed by the computer.

Object code is also called binary code and machine code. Once a program is compiled, a line of source code is no longer meaningful in the execution of the program.

When you use an interpreter, it must be present each time you wish to run your program. For example, in BASIC you have to execute the BASIC interpreter first and then load your program and type RUN each time you want to use it. The BASIC interpreter then examines your program one line at a time for correctness and then executes it. This slow process occurs every time the program runs. By contrast a compiler converts your program into object code that can be directly executed by your computer. Because the compiler translates your program only once, all you need to do is execute your program directly, usually by the simple process of typing its name. Thus compilation is one-time cost, while interpreted code incurs an overhead cost each time a program runs.

# DATA TYPES

## Basic Data Types

There are five atomic data types in C: character, integer, floating point, double floating point, and valueless. The sizes of these types are shown in below Table.

| Type | Bit Width | Range |
|---|---|---|
| char | 8 | 0 to 255 |
| int | 16 | -32768 to 32767 |
| float | 32 | 3.4E - 38 to 3.4E + 38 |
| double | 64 | 1.7E - 308 to 1.7E + 308 |
| void | 0 | valueless |

***Size and Range of C Basic Data Types***

Values of type **char** are used to hold ASCII characters or any 8-bit quantity. Variables of type **int** are used to hold integer quantities. Variables of type **float** and **double** are used to hold real numbers.

The **void** type has three uses. The first is to declare explicitly a function a returning no value; the second is to declare explicitly a function as having no parameters; the third is to create generic pointers.

## Type Modifiers

Expecting type void, the basic data types may have various modifiers preceding them. A modifier is used to alter the meaning of the base type to fit the needs of various situations more precisely.

The modifiers **signed**, **unsigned**, **long**, and **short** may be applied to character and integer base types. However, **long** may also be applied to **double**. Table below shows all allowed combinations that adhere to the ANSI C standard, along with their bit widths and range assuming a 16-bit word.

The use of **signed** on integers is redundant (but allowed) because the default integer declaration assumes a signed number.

The difference between signed and unsigned integers is in the way the high-order bit of the integer is interpreted. If a signed integer is specified, then the compiler will generate code that assumes the high-order bit of an integer is to be used as a sign flag. If the sign flag is 0, then the number is positive; if it is 1, then the number is negative.

| Type | Bit Width | Range |
|---|---|---|
| char | 8 | -128 to 127 |
| unsigned char | 8 | 0 to 255 |
| signed char | 8 | -128 to 127 |
| int | 16 | -32768 to 32767 |
| unsigned int | 16 | 0 to 65535 |
| signed int | 16 | -32768 to 32767 |
| short int | 16 | -32768 to 32767 |
| unsigned short int | 16 | 0 to 65535 |
| signed short int | 16 | -32768 to 32767 |
| long int | 32 | -2147483648 to 2147483647 |
| signed long int | 32 | -2147483648 to 2147483647 |
| float | 32 | 3.4E - 38 to 3.4E + 38 |
| double | 64 | 1.7E - 308 to 1.7E + 308 |
| long double | 64 | 1.7E - 308 to 1.7E + 308 |

***All possible Combinations of C Basic Types and Modifiers***

### Access Modifiers

C has two type modifiers that are used to control the ways in which variables may be accessed or modified. These modifiers are called **const** and **volatile**.

Variables of type **const** may not be changed during execution by your program. For example: `const int a;`

Will create an integer variable called a that cannot be modified by your program.

The modifier **volatile** is used to tell the compiler that a variable's value can be changed in ways not explicitly specified by the program. For example, a global variable's address can be passed to the clock routine of the operating system and used to hold the real-time of the system.

## DECLARATION OF VARIABLES

All variables must be declared before they are used. There are two types of variable declaration in C, local and global.

### Local Variables

Variables that are declared inside a function are called local variables. One of the most important things to understand about local variables is that they exist only while the block of code in which they are declared is executing. That is, a local variable is created upon entry into its block and destroyed upon exit.

The most common code block in which local variables are declared is the function.

The use of **signed** on integers is redundant (but allowed) because the default integer declaration assumes a signed number.

The difference between signed and unsigned integers is in the way the high-order bit of the integer is interpreted. If a signed integer is specified, then the compiler will generate code that assumes the high-order bit of an integer is to be used as a sign flag. If the sign flag is 0, then the number is positive; if it is 1, then the number is negative.

| Type | Bit Width | Range |
|---|---|---|
| char | 8 | -128 to 127 |
| unsigned char | 8 | 0 to 255 |
| signed char | 8 | -128 to 127 |
| int | 16 | -32768 to 32767 |
| unsigned int | 16 | 0 to 65535 |
| signed int | 16 | -32768 to 32767 |
| short int | 16 | -32768 to 32767 |
| unsigned short int | 16 | 0 to 65535 |
| signed short int | 16 | -32768 to 32767 |
| long int | 32 | -2147483648 to 2147483647 |
| signed long int | 32 | -2147483648 to 2147483647 |
| float | 32 | 3.4E - 38 to 3.4E + 38 |
| double | 64 | 1.7E - 308 to 1.7E + 308 |
| long double | 64 | 1.7E - 308 to 1.7E + 308 |

***All possible Combinations of C Basic Types and Modifiers***

### Access Modifiers

C has two type modifiers that are used to control the ways in which variables may be accessed or modified. These modifiers are called **const** and **volatile**.

Variables of type **const** may not be changed during execution by your program. For example: `const int a;`

Will create an integer variable called a that cannot be modified by your program.

The modifier **volatile** is used to tell the compiler that a variable's value can be changed in ways not explicitly specified by the program. For example, a global variable's address can be passed to the clock routine of the operating system and used to hold the real-time of the system.

## DECLARATION OF VARIABLES

All variables must be declared before they are used. There are two types of variable declaration in C, local and global.

### Local Variables

Variables that are declared inside a function are called local variables. One of the most important things to understand about local variables is that they exist only while the block of code in which they are declared is executing. That is, a local variable is created upon entry into its block and destroyed upon exit.

The most common code block in which local variables are declared is the function.

## Logical Operators

| Operator | Action |
|---|---|
| && | AND |
| \|\| | OR |

*Logical Operators*

## Relational Operators

| Operator | Action |
|---|---|
| > | Greater than |
| >= | Greater than or equal |
| < | Less than |
| <= | Less than or equal |
| == | Equal |
| != | Not equal |

*Relational Operators*

## Bitwise Operators

| Operator | Action |
|---|---|
| & | AND |
| \| | OR |
| ^ | Exclusive OR (XOR) |
| ~ | One's complement |
| >> | Shift right |

*Bitwise Operators*

## The ? operator

C has a very powerful and convenient operator that can be used to replace certain statements of the if-then-else form. The ternary operator ? takes the general form

```
Exp1 ? Exp2 : Exp2
```

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon. The ? operator works like this. Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the expression. If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression. For example

```
x = 10;

y = X>9 ? 100 : 200;
```

In this example, y will be assigned the value 100. If x had been less than or equal to 9, y would have received the value 200.

# Program Control Statements

## Conditional Statements

C supports two types of conditional statements: if and switch. In addition the ? operator is an alternative to the if in certain circumstances.

### If

The general form of the if statement is

```
if(expression) statement;
        else statement;
```

where *statement* may be either a single statement or a block of statements.
The general form of the **if** with blocks of statement is

```
if(expression) {
        statement sequence
}
else {
        statement sequence
}
```

If the expression is true (anything other than 0), the statement or block that forms the target of the **if** is executed; otherwise, the statement or block that is target of the **else** is executed. Remember, only the code associated with the **if** or the code that is associated with the **else** executes, never both.

### Nested Ifs

One of the most confusing aspects of if statements in any programming language is nested ifs. A *nested if* is an **if** statement that is the object of either an **if** or **else**. The reason that nested **if**s are so troublesome is that it can difficult to know what else associates with what if.

```
if(x)
        if(y) printf("1");
        else printf("2");
```

To which if does the **else** refer?
Fortunately, C provides a very simple rule for resolving this type of situation. In C, the **else** is linked to the closest preceding **if** that does not already have an **else** statement associated with it. In this case, the else is associated with the **if**(y) statement. To make the **else** associated, as with the **if**(x) you must use braces to override its normal association, as shown here.

```
if(x) {
        if(y) printf("1");
}
else printf("2");
```

The **else** is now associated with the **if**(x) because it is no longer part of the **if**(y) object block. Because of C's scope rules, the **else** now has no knowledge of the **if**(y) statement because they are no longer in the same block of code.

## Switch

C has a built-in multiple branch decision statement called **switch**. A variable is successively tested against a list of integer or character constants. When a match is found, a statement or block of statements is executed. The general form of the **switch** statement is

```
switch(variable) {
        case constant1:
                statement sequence
                break;
        case constant2:
                statement sequence
                break;
        .
        .
        .
        default:
                statement sequence
}
```

where the **default** statement is executed if no matches are found. The **default** is optional and, if not present, no action takes place if all matches fail. When a match is found, the statement associated with that **case** is executed until the **break** statement is reached or, in the case of the default (or last **case** if no **default** is present), the end of the switch statement is encountered.

## Loops

In C, and all other modern programming languages, loops allow a set of instructions to be performed until a certain condition is reached. This condition may be predefined as in the **for** loop, or open-ended as in the **while** and **do** loops.

## for

The general format of C's **for** loop is probably familiar to you because it is found in one form or another in all procedural programming languages. However, in C it has unexpected flexibility and power.
The general form of the **for** statement is

```
for(initliazation; condition; increment) statement;
```

The for statement allows many variants, but there are three main parts:

1. The initialization is usually an assignment statement that is used to set the loop control variable.

2. The condition is a relational expression that determines when the loop will exit.

3. The increment defines how the loop control variable will change each time the loop is repeated.

These three major sections must be separated by semicolons. The **for** loop continues to execute as long as the condition is true. Once the condition becomes false, program execution resumes on the statement following the **for** loop.

## while

The second loop available in C is the while. The general form is

while(*condition*) *statement*;

where *statement*, as stated earlier, is either any empty statement, a single statement, or a block of statements that is to be repeated. The condition may be any expression, with true being any nonzero value, The loop iterates while the condition is true. When the condition becomes false, program control passes to the line after the loop code.
The following example shows a keyboard input routine that simply loop until A is presses.

```
void wait_for_char(void)
{
      char ch;

      ch = '\0';
      while(ch! != 'A') ch = getchar();
}
```

## do/while

Unlike the for and while loops that test the loop condition at the top of the loop, the do/while loops checks its condition at the bottom of the loop. This means that a do/while loop always executes at least once. The general form of the do/while loop is

```
do{
      statement sequence;
} while(condition);
```

Although the braces are not necessary when only one statement is present, they are usually used to improve readability and avoid confusion with the while.
This do/while reads numbers from the keyboard until one is less then or equal to 100.

```
do {
      scanf("%d", &num);
} while(num > 100);
```

### Break

The **break** statement has two uses. The first is to terminate a case in the switch statement. The second us is to force immediate termination of a loop, bypassing the normal loop conditional test.

### exit()

The function **exit()**, which is found in the standard library, causes immediate termination of the entire program, Because the **exit()** function stops program execution and forces a return to the operating system, its use is somewhat specific as a program control device.

### Continue

The **continue** statement works somewhat like the break statement. But, instead of forcing termination, **continue** forces the next iteration of the loop to take place, skipping any code in between.

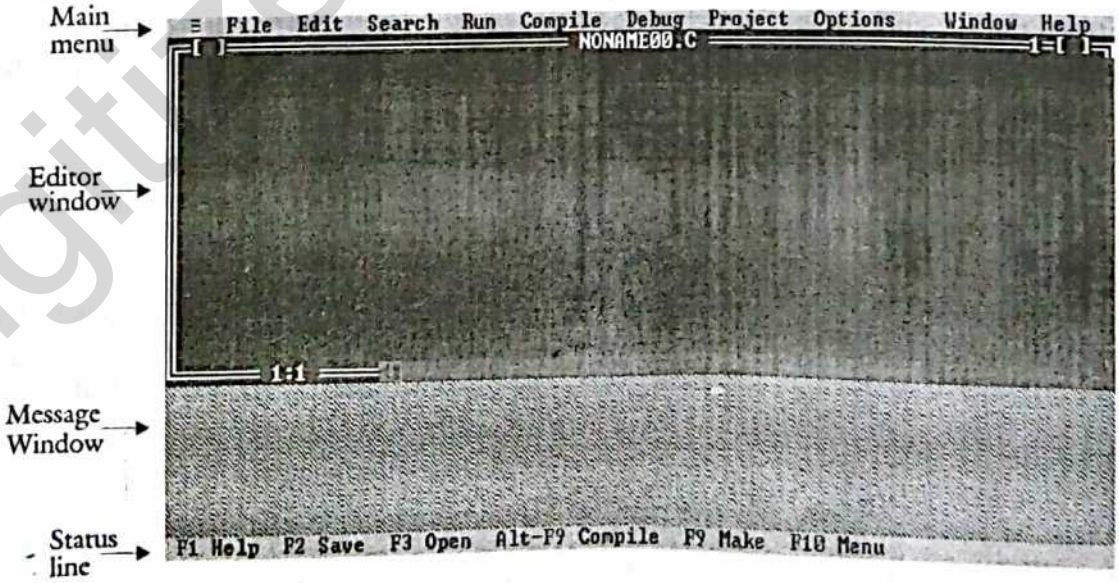
## Turbo C Integrated Development Environment

Turbo C has two separate mode of operation. The first is called the integrated development environment (IDE). Using the IDE, editing, compiling, and executing programs are controlled by single keystrokes and easy-to-use, intuitive menus. The second method of operation involves the traditional approach where you first use an editor to create a program source file, then compile, link, and run the program from the command line.

### Executing the Turbo C IDE

To execute the integrated environment, simply type "TC" on the command line followed by a carriage return. When Turbo C begins execution you see the screen shown below. It consists of these four parts, in order from top to bottom.

* The main menu
* The editor window
* The message window
* The status line



### Using the Mouse

The Turbo C IDE can be operated using either the keyboard or the mouse. Although a mouse is not required, mouse support has been carefully integrated into the Turbo C IDE, and a mouse is certainly an excellent addition.

### The Main Menu

To activate the main menu, press the F10 key. When you do this, one of the menu items becomes highlighted.

The main menu is used either to tell Turbo C to do something, such as load a file or compile a program, or to set an option. Once the main menu is activated, there are to ways to make a main menu selection using the keyboard. First, you can use the arrow keys to highlight the item you want and then press Enter. Second, you can simply type the first letter of the desired menu item. For example, to select Edit, you type an E. You can enter the letters in either upper or lowercase. if you have a mouse, you simply click on the main menu item that you want to activate.

### Dialog Boxes

If a pull-down menu item is followed by three periods, selecting the item displays a dialog box. Dialog boxes allow input that is not easily accomplished using a menu. Dialog boxes consist of one or more of the following items:

Action buttons
Check boxes
Input boxes
List boxes
Radio buttons

### Using the IDE Compiler

Press Alt + C from the keyboard, a pull down menu Compile will be displayed, select the option compile from the menu by using cursor keys and press Enter key. Turbo C IDE will invoke compiler to compile your program.

If you want to run your program, press Alt + R and select Run option from the pull down menu or press Ctrl + F9 to execute your program. If you have already compiled your program Turbo C will execute immediate your program otherwise it will first compile and then execute your program.

## Using the Command-Line Compiler

Unlike Turbo C's IDE, which provides a complete development environment, Turbo C command-line compiler performs only two tasks: it compiles and links your C program. In this case, you use your own text editor to create a program and then use the command-line compiler to translate it into executable code. This procedure represents the traditional method of compilation and linking. The main reason that you might want to use the command-line compiler instead of the IDE is if you have a favorite text editor that you want to use. Whatever the reason, if you choose to use the command-line version of Turbo C, you will be happy to know that it compiles your program in the same way that the compiler inside the IDE does.

The name of the command-line compiler is TCC.EXE. The simplest way to compile a program using the command-line version of Turbo C is to use this general form:

```
C> TCC program-name
```

For example, if you want to compile a program called MYPROG.C, type the following at the DOS prompt;

```
C> TCC MYPROG.C
```

Assuming that there are no errors in the program, this compiles and links MYPROG.C with the proper library files.
Like the IDE compiler, the command-line compiler allows you to control its exact operation using command-line options. All compiler/linker options come before the file name on the command line. Also, all options begin with a dash (minus sign). Generally, following an option with a dash turns that option off. Keep in mind that the options are case-sensitive.

# Practical 1

## Write a program to print 'hello' on the screen.

### Program

```
#include <stdio.h>

int main(void)
{
      printf("Hello");

      return 0;
}
```

### Output

```
Hello
```

## Practical 2

**Write a program to pint your name and address. Use one printf with a new line character in it.**

### Program

```
#include <stdio.h>

int main(void)
{
      printf("Ahmad Kamal\n14-F Gulberg III, Lahore");

      return 0;
}
```

### Output

```
Ahmad Kamal
14-F Gulberg III, Lahore
```

## Practical 3

**Write a program which prints the minimum and maximum values of an integer.**

### Program

```
#include <stdio.h>
#include <limits.h>

int main(void)
{
      printf("minimum int = %i, ", INT_MIN);
      printf("maximum int = %i\n ", INT_MAX);

      return 0;
}
```

### Output

```
minimum int = -32768, maximum int = 32767
```

# Practical 4

## Write a program to compute the area of a rectangle.

### Program

```
#include <stdio.h>

int main(void)
{
        int width, length, area;
        width = 15;
        length = 40;

        area = width * length;

        printf("Rectangale width is %d\n", width);
        printf("Rectangale length is %d\n", length);
        printf("Area of rectangle is %d\n", area);

        return 0;
}
```

### Output

```
Rectangle width is 15
Rectangle length is 40
Area of rectangle if 600
```

## Practical 5

**Write a program which prints atext of 4 lines consisting of characters, integer values and floating point values using printf statement.**

### Program

```
#include <stdio.h>

int main(void)
{
      float north1 = 23.30, north2 = 36.45;
      int east1 = 61;
      float east2 = 75.30;

      printf("Pakistan is located between %f Degrees and\n", north1);
      printf("%f Degrees North latitude and %d\n", north2, east1);
      printf("Degrees and %f Degrees East longitude in the\n", east2);
      printf("northern hemisphere in South-East Asia.\n");

      return 0;
}
```

### Output

```
Pakistan is located between 23.30 Degrees and
36.45 Degrees North latitude and 61
Degrees and 75.30 Degrees East longitude in the
northern hemisphere in South-East Asia.
```

## Practical 6

**Write a program to read in 3 integers and print their sum.**

### Program

```
#Include <stdio.h>

int main(void)
{
      int value1, value2, value3, sumall;

      printf("Enter first value: ");
      scanf("%d", &value1);
      printf("\nEnter second value: ");
      scanf("%d", &value2);
      printf("\nEnter third value: ");
      scanf("%d", &value3);

      sumall = value1 + value2 + value3;
      printf("\n\nSum of 3 integers = %d", sumall);

      return 0;
}
```

### Output

```
Enter first value: 15
Enter second value: 25
Enter third value: 11

Sum of 3 integers = 51
```

# Practical 7

## Write a program to read in two integers and display one as a percentage of the other.

### Program

```
#include <stdio.h>

int main(void)
{
      int value1, value2, percent;

      printf("Enter first value: ");
      scanf("%d", &value1);
      printf("\nEnter second value: ");
      scanf("%d", &value2);

      percent=(value1 * 100) / value2;
      printf("\n%d is %d%% of %d", value1, percent, value2);

      return 0;
}
```

### Output

```
Enter first value: 20
Enter second value: 40
20 is 50% of 40
```

## Practical 8

**Write a program that reads and prints using Escape Sequence. (Asking the name, age, height, gender of the students using scanf and printf statements.**

### Program

```
#include <stdio.h>

int main(void)
{
	char name[20];
	int age;
	float height;
	char gender[6];

	printf("Enter your name: ");
	scanf("%s", &name);
	printf("\nEnter your age: ");
	scanf("%d", &age);
	printf("\nEnter your height: ");
	scanf("%f", &height);
	printf("\nEnter your gender: ");
	scanf("%s", &gender);

	printf("\n\nYour name is %s. ", name);
	printf("You are %d years old.", age);
	printf("\nYour height is %f feet. ", height);
	printf("You are a %s person.", gender);

	return 0;
}
```

### Output

```
Enter your name: Ahmad
Enter your age: 18
Enter your height: 5.8
Enter your gender: Male

Your name is Ahmad. You are 18 years old.
Your height is 5.8 feet. You are a Male person.
```

## Practical 9

**Write a program to calculate the area of a circle.**

### Program

```
#include <stdio.h>

int main(void)
{
        const float pi = 3.1415926;
        float radius, area;

        printf("What is the radius of a circle? ");
        scanf("%f", &radius);

        area = pi * radius * radius;
        printf("\n\nArea of a circle is %f", area);

        return 0;
}
```

### Output

```
What is the radius of a circle? 180

Area of a circle is 101787.59
```

# Practical 10

**Write a program to read in two integers and display which one is the largest.**

## Program

```
#include <stdio.h>

int main(void)
{
      int x, y;

      printf("Enter first value : ");
      scanf("%d", &x);
      printf("\nEnter second value : ");
      scanf("%d", &y);

      if(x > y)
            printf("\n\n%d is the largest integer value", x);
      else
            printf("\n\n%d is the largest integer value", y);

      return 0;
}
```

## Output

```
Enter first value : 10
Enter second value : 15

15 is the largest integer value
```

# Practical 11

**Write a program which uses operators (calculate the are of triangles, volume of sphere and arrange the resultant values in ascending order).**

## Program

```
#include <stdio.h>
#include <math.h>

int main(void)
{
      float a, b, c, s, area, d, vol;

      printf("Enter triangle side A : ");
      scanf("%f", &a);
      printf("\nEnter triangle side B : ");
      scanf("%f", &b);
      printf("\nEnter triangle side C : ");
      scanf("%f", &c);
      printf("\nEnter diameter of sphere : ");
      scanf("%f", &d);

      s =(a + b + c) / 2;
      area = sqrt(s * (s - a) * (s - b) * (s - c));

      vol = (3.14 / 6) * (d * d * d);

      if(vol > area) {
            printf("\nArea of triangle = %f", area);
            printf("\nVolume of sphere = %f", vol);
      }
      else {
            printf("\nVolume of sphere = %f", vol);
            printf("\nArea of triangle = %f", area);
      }

      return 0;
}
```

## Output

```
Enter triangle side A : 10
Enter triangle side B : 10
Enter triangle side C : 15
Enter diameter of sphere : 15

Area of triangle = 49.60
Volume of sphere = 1766.25
```

Practical

# 13

## Write a program to print a simple triangle of asterisks.

### Program

```
#include <stdio.h>

int main(void)
{
      int i, j;

      for(i = 1; i <= 10; i = i + 1) {
            for(j = 1; j <= i; j = j + 1)
                  printf("*");
      printf("\n");
      }

      return 0;
}
```

### Output

```
*
**
***
****
*****
******
*******
********
*********
**********
```

# Practical 13

## Write a program to print a simple triangle of asterisks.

### Program

```
#include <stdio.h>

int main(void)
{
        int i, j;

        for(i = 1; i <= 10; i = i + 1) {
                for(j = 1; j <= i; j = j + 1)
                        printf("*");
        printf("\n");
        }

        return 0;
}
```

### Output

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Practical

# 14

**Write a program which uses 'for' loop statement (generate the multiplication table from 2 to 20).**

## Program

```
#include <stdio.h>

int main(void)
{
        int x, y, tnum;

        for(x = 2; x <= 20; x++) {
                for(y = 1; y <= 10; y++) {
                        tnum = x * y;
                        printf("%d * %d = %d\n", x, y, tnum);
                }
        }

        return 0;
}
```

## Output

```
2 * 1 = 2
2 * 2 = 4
..
..
2 * 10 = 20
3 * 1 = 3
..
..
9 * 10 = 90
..
..
20 * 10 = 200
```

# Practical 15

**Write a program to print the numbers between 1 and 10, along with an indication of whether each is even or odd.**

## Program

```
#include <stdio.h>

int main(void)
{
      int i;

      for(i = 1; i <= 10; i = i + 1)
            {
            if(i % 2 == 0)
                  printf("%d is even\n", i);
            else  printf("%d is odd\n", i);
            }

       return 0;
}
```

## Output

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
8 is even
9 is odd
10 is even
```

# Practical 16

**Write a program to print even numbers from 20 to 40 and find sum of all numbers using 'while'loop statement.**

## Program

```
#include <stdio.h>

int main(void)
{
      int x = 20, sum = 0;

      while(x <= 40) {
            printf("%d\n", x);
            sum += x;
            x += 1;
      }

      printf("Sum from 20 to 40 = %d", sum);

      return 0;
}
```

## Output

```
20
22
24
26
28
30
32
34
36
38
40
Sum from 20 to 40 = 330
```

# Practical 17

**Write a program which uses 'while' loop and nested 'while' loop (use 'for' loop and continue the process in 'while' loop satisfying this condition.**

## Program

```
#include <stdio.h>

int main(void)
{
      int x, loop, count;

      for(x = 1; x <= 2; x++) {
            loop = 1;

            while(loop <= 5) {
                  count = 1;

                  while(count <= loop) {
                        printf("%d", count);
                        count++;
                  }
                  printf("\n");

                  loop++;
            }
      }

      return 0;
}
```

## Output

```
1
12
123
1234
12345
1
12
123
1234
12345
```

# Practical 18

## Finding the factorial of N using 'while' loop, read value of N using scanf, and print factorial of various N.

### Program

```
#include <stdio.h>

int main(void)
{
	int n, result = 1, counter = 2;

	printf("Enter the value of N : ");
	scanf("%d", &n);

	while(counter <= n) {
		result = counter * result;
		counter += 1;
	}

	printf("\nFactorial of N is %d", result);

	return 0;
}
```

### Output

```
Enter value of N : 10

Factorial of N is 24320
```

# Practical 19

**Draw a checkerboard and print it using if-else statement, and extend the program using nested if-else.**

## Program

```
#include <stdio.h>

int main(void)
{
	int board, i, j, k;
	board =0;
	k = 0;

	printf("How many biff do you want the square : ");
	scanf("%d", &board);
	printf("\n");

	for(i = 0; i<board; i++){
		for(j = 0; j<board; j++){
			if(k == 0){
				printf("_");
				k = 1;
			}
			else{
				if(k == 1){
					printf("#");
					k = 0;
				}
			}
		}
		printf("\n");
	}

	return 0;
}
```

## Output

```
How many biff do you want the square : 10
_#_#_#_#_#
_#_#_#_#_#
_#_#_#_#_#
_#_#_#_#_#
_#_#_#_#_#
_#_#_#_#_#
_#_#_#_#_#
_#_#_#_#_#
_#_#_#_#_#
_#_#_#_#_#
```

# Practical 20

**Write a program to read an integer value and print out the price of different fruits using switch statement.**

## Program

```
#include <stdio.h>

int main(void)
{
      int x;

      printf("What fruit price you want to see?");
      printf("\n(1) Oranges, (2) Apples");
      printf("\n(3) Bananas, (4) Cherries;
      printf("\nEnter your choice : ");
      scanf("%d", &x);

      switch (x) {
            case 1:
                  printf("\nOranges are 20 per KG");
                  break;
            case 2:
                  printf("\nApples are 40 per KG");
                  break;
            case 3:
                  printf("\nBananas are 15 per KG");
                  break;
            case 4:
                  printf("\nCherries are 150 per KG");
                  break;
            default:
                  printf("\nSorry, we are out of stock");

      }

      return 0;
}
```

## Output

```
What fruit price you want to see?
(1) Oranges, (2) Apples
(3) Bananas, (4) Cherries
Enter your choice : 1

Oranges are 20 per KG
```

# Practical 21

**Write a program which uses a 'switch' statement and breaks the program if certain condition is observed. Repeat the program with case statement.**

## Program

```
#include <stdio.h>

int main(void)
{
      int month;

      do {
            printf("\nType the number of a month : ");
            scanf( "%d", &month );

            printf("\n\nName of month %d is ", month);

            switch (month) {
                  case 1: printf("January"); break;
                  case 2: printf("February"); break;
                  case 3: printf("March"); break;
                  case 4: printf("April"); break;
                  case 5: printf("May"); break;
                  case 6: printf("June"); break;
                  case 7: printf("July"); break;
                  case 8: printf("August"); break;
                  case 9: printf("September"); break;
                  case 10: printf("October"); break;
                  case 11: printf("November"); break;
                  case 12: printf("Devember"); break;
                  default: printf("Unknown"); continue;
            }
      } while (month < 1 || month > 12);

      return 0;
}
```

## Output

```
Type the number of a month : 8

Name of month 8 is August
```

# Practical 22

**Write a function celsius() to convert degrees Fahrenheit to degrees Celsius.**

## Program

```
#include <stdio.h>

float celsius(int x);

int main(void)
{
      int fah;
      float cel;

      printf("\nEnter temperature in Fahrenheit : ");
      scanf("%d", &fah);

      cel = celsius(fah);
      printf("\nTemperature in Celsius is %f", cel);

      return 0;
}

float celsius(int x)
{
      float y;
      y = 5 * (x - 32.) /9;
      return y;
}
```

## Output

```
Enter temperature in Fahrenheit : 122
Temperature in Celsius is 50
```

Practical

# 23

**Write a function, which generates factorial of N and calls this function in the 'main' program.**

## Program

```
#include <stdio.h>

int factorial(int x);

int main(void)
{
      int n, f;

      printf("Enter the value of N : ");
      scanf("%d", &n);

      f = factorial(n);
      printf("\nFactorial of N is %d", f);

      return 0;
}


int factorial(int x)
{
      int i;
      int fact = 1;
      for(i = 2; i <= x; i = i + 1)
            fact = fact * i;
      return fact;
}
```

## Output

```
Enter the value of N : 10

Factorial of N is 24320
```

# Practical 24

**Write a program which uses multiple arguments in a function (develop a user-defined function to generate a rectangle. Use the function for passing arguments to draw different sizes of rectangles and squares.**

## Program

```
#include <stdio.h>

void box(int length, int width);
int main(void)
{
      box(5,5);
      box(5,10);
      box(7,7);
      box(3,8);
      return 0;
}

void box(int length, int width)
{
      int a, b, x;
      for(a = 1; a <= width; a++)
            printf("*");
      for(b = 1; b <= length; b++) {
            x=0;
            printf("\n*");
            while(x <= (width - 3)) {
                  x=x+1;
                  printf(" ");
            }
            printf("*");
      }
      printf("\n");
      for(a = 1; a <= width; a++)
            printf("*");
      printf("\n");
}
```

## Output

```
*****    **********
*   *    *         *
*   *    *         *
*   *    *         *
*****    **********
*******  ********
*     *  *       *
*     *  ********
*     *
*     *
*     *
*******
```